SANDIA REPORT

SAND2014-18874
Unlimited Release
Printed October 2014

MueLu User's Guide for Trilinos Version 11.12

Andrey Prokopenko, Jonathan J. Hu, Tobias A. Wiesner, Christopher M. Siefert, Raymond S. Tuminaro

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2014-18874
Unlimited Release
Printed October 2014

MueLu User's Guide for Trilinos Version 11.12

Andrey Prokopenko
Scalable Algorithms
Sandia National Laboratories
Mailstop 1318
P.O. Box 5800
Albuquerque, NM 87185-1318
aprokop@sandia.gov

Tobias Wiesner
Institute for Computational Mechanics
Technische Universität München
Boltzmanstraße 15
85747 Garching, Germany
wiesner@lnm.mw.tum.de

Jonathan J. Hu
Scalable Algorithms
Sandia National Laboratories
Mailstop 9159
P.O. Box 0969
Livermore, CA 94551-0969
jhu@sandia.gov

Christopher M. Siefert
Computational Multiphysics
Sandia National Laboratories
Mailstop 1322
P.O. Box 5800
Albuquerque, NM 87185-1322
csiefer@sandia.gov

Raymond S. Tuminaro
Computational Mathematics
Sandia National Laboratories
Mailstop 9159
P.O. Box 0969
Livermore, CA 94551-0969
rstumin@sandia.gov

Abstract

This is the official user guide for the MUELU multigrid library in Trilinos version 11.12. This guide provides an overview of MUELU, its capabilities, and instructions for new users who want to start using MUELU with a minimum of effort. Detailed information is given on how to drive MUELU through its XML interface. Links to more advanced use cases are given. This guide gives information on how to achieve good parallel performance, as well as how to introduce new algorithms. Finally, readers will find a comprehensive listing of available MUELU options. *Any options not documented in this manual should be considered strictly experimental.*

Acknowledgment

Many people have helped develop MUELU and/or provided valuable feedback, and we would like to acknowledge their contributions here: Tom Benson, Julian Cortial, Stefan Domino, Travis Fisher, Jeremie Gaidamour, Axel Gerstenberger, Chetan Jhurani, Mark Hoemmen, Jonathan Hu, Paul Lin, Eric Phipps, Andrey Prokopenko, Siva Rajamanickam, Chris Siefert, Paul Tsuji, Ray Tuminaro, and Tobias Wiesner.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This guide is gives an overview of MUELU's capabilities. If you are looking for a tutorial, please refer to the MUELU tutorial in `muelu/doc/Tutorial`. New users should start with §2. It strives to give the new user all the information he/she might need to begin using MUELU quickly. Those who are interested in optimizing parallel performance should refer to section §3. Users who simply need to look up particular options should refer to the complete set of supported options is given in §4. Power users or developers who are interested in extending MUELU should read §**??**, which describes how MUELU can be modified to incorporate new algorithms.

If you find errors or omissions in this guide, please contact the MUELU developer list, `muelu-developers@software.sandia.gov`.

Chapter 2

Getting Started

This section is meant to get you using MUELU as quickly as possible.

2.1 Prerequisites

It's assumed the reader is comfortable with TEUCHOS referenced-counted pointers (RCPs) for memory management. An introduction to RCPs can be found in [3]. This guide also assumes familiarity with the `Teuchos::ParameterList` class [11].

2.2 Overview of MUELU

MUELU is an extensible multigrid library that is part of the TRILINOS project. MUELU works with EPETRA (32- and 64-bit versions) and TPETRA matrix types. The library is written in C++ and allows for different ordinal (index) and scalar types. MUELU is designed to be efficient on many different computer architectures, from workstations to supercomputers. While it is MPI based, MUELU is relies on the "MPI+X" principle, where "X" can be threading or CUDA.

MUELU provides a number of different multigrid algorithms:

1. smoothed aggregation algebraic multigrid (AMG), appropriate for Poisson-like and elasticity problems
2. Petrov-Galerkin aggregation AMG for convection-diffusion problems
3. aggregation-based AMG for problems arising from the eddy current formulation of Maxwell's equations

MUELU's software design allows for the rapid introduction of new multigrid algorithms.

2.3 Quick Start

The MUELU C++ interface works with either EPETRA or TPETRA matrices. Solver options can be provided either by XML input files or parameter lists (key/value pairs).

In this example for TPETRA users, options are read from an XML text file.

```
Teuchos::RCP<Tpetra::CrsMatrix<> > A;
// create A here ...
Teuchos::RCP<MueLu::TpetraOperator> mueLuPreconditioner;
std::string optionsFile = "mueluOptions.xml";
mueLuPreconditioner = MueLu::CreateTpetraPreconditioner(A, optionsFile);
```

A similar interface exists for EPETRA users.

```
Teuchos::RCP<Epetra_CrsMatrix> A;
// create A here ...
Teuchos::RCP<MueLu::EpetraOperator> mueLuPreconditioner;
std::string optionsFile = "mueluOptions.xml";
mueLuPreconditioner = MueLu::CreateEpetraPreconditioner(A, optionsFile);
```

In this example for TPETRA users, options are provided via a `Teuchos::ParameterList`.

```
Tpetra::CrsMatrix<> A;
// create A here ...
Teuchos::RCP<MueLu::TpetraOperator> mueLuPreconditioner;
Teuchos::ParameterList paramList;
paramList.set("verbosity", "medium");
paramList.set("multigrid algorithm", "sa");
paramList.set("aggregation: type", "uncoupled");
paramList.set("smoother: type", "CHEBYSHEV");
paramList.set("coarse: max size", 500);
mueLuPreconditioner = MueLu::CreateTpetraPreconditioner(A, paramList);
```

2.4 Multigrid Introduction

A brief multigrid description is given here (see [5] or [12] for more information). A multigrid solver tries to approximate the original problem of interest with a sequence of smaller (*coarser*) problems. The solutions from the coarser problems are interpolated and combined in order to accelerate convergence of the original (*fine*) problem. on the finest grid. A simple multilevel iteration is illustrated in Figure 2.1. (This algorithm is borrowed from [8].)

In the multigrid iteration in Figure 2.1, the $S_k^1()$'s and $S_k^2()$'s are called *pre-smoothers* and *post-smoothers*. They are approximate solvers (e.g. symmetric Gauss-Seidel), and the subscript k

function MULTILEVEL(A_k, b, u, k)
  // Solve A_k u = b (k is current grid level)
  $u = S_k^1(A_k, b, u)$
  if ($k \neq$ **Nlevel** $-$ **1**) **then**
    $P_k =$ determine_interpolant(A_k)
    $\hat{r} = P_k^T(b - A_k u)$
    $\hat{A}_{k+1} = P_k^T A_k P_k$
    $v = 0$
    multilevel($\hat{A}_{k+1}, \hat{r}, v, k+1$)
    $u = u + P_k v$
    $u = S_k^2(A_k, b, u)$
  end if
end function

Figure 2.1. High level multigrid V cycle consisting of 'Nlevel' levels to solve $Ax = b$, with $A_0 = A$.

denotes the number of applications of the approximate solution method. The purpose of a smoother is to quickly reduce certain error modes in the approximate solution on a level i. For symmetric fine level problems, the pre- and post-smoothers must be chosen to maintain symmetry (e.g., forward Gauss-Seidel for the pre-smoother and backward Gauss-Seidel for the post-smoother). For the coarsest level, often a direct solve is employed if the problem is small enough. The P_k's are interpolation matrices that transfer solutions from coarse levels to finer levels. In geometric multigrid, the P_k's are determined by the application, whereas they are automatically generated in an algebraic multigrid method. For symmetric problems, typically $R_k = P_k^T$. For nonsymmetric problems, this is not necessarily true. The A_k's are the coarse level problems and are generated through a so-called Galerkin product.

Note that the algebraic multigrid algorithms implemented in MUELU generate the grid transfers P_k automatically and the coarse problems A_k via a sparse triple matrix product. There are many smoothers and direct solvers available for use in MUELU through the IFPACK, IFPACK2, AMESOS, and AMESOS2 packages (see §4).

2.5 Configuring and Building

MUELU has been compiled successfully with GNU (many 4.x versions), Intel 12.1/13.1 and clang 3.4 C++ compilers.

2.5.1 Required Dependencies

MUELU requires that TEUCHOS and either EPETRA/IFPACK or TPETRA/IFPACK2 be enabled.

2.5.2 Recommended Dependencies

We strongly recommend that you enable the following dependencies along with MUELU:

- EPETRA stack: AZTECOO, EPETRA, AMESOS, IFPACK, ISORROPIA, GALERI, ZOLTAN
- TPETRA stack: AMESOS2, BELOS, GALERI, IFPACK2, TPETRA, ZOLTAN2

2.5.3 Tutorial Dependencies

In order to run the MUELU Tutorial [13] located in `muelu/doc/Tutorial`, MUELU must be configured with the following dependencies enabled:

AZTECOO, AMESOS, AMESOS2, BELOS, EPETRA, IFPACK, IFPACK2, ISORROPIA, GALERI, TPETRA, ZOLTAN, ZOLTAN2.

2.5.4 Complete List of Direct Dependencies

Table 2.1 enumerates the dependencies of MUELU. Certain dependencies are optional, whereas others are required. Furthermore, MUELU's tests depend on certain libraries that are not required if you only want to link against the MUELU library and do not want to compile its tests.

AMESOS2 is necessary if you want to use a sparse direct solve on the coarsest level. ZOLTAN2 is necessary if you want to be able to rebalance a matrix in parallel (see §3). BELOS is necessary if you want to be able to use MUELU as a preconditioner instead of a solver.

☛ Note that MUELU has also been successfully tested with SuperLU 4.1 and SuperLU 4.2.

☛ Be aware that other packages such as ZOLTAN and ZOLTAN2 may come with additional requirements for third party libraries (such as ParMetis), which are not listed here as explicit dependencies of MUELU. It is highly recommended to install ParMetis 3.1.1 or newer for ZOLTAN and ParMetis 4.0.x for ZOLTAN2.

2.5.5 Configuring

You should configure and build MUELU in a directory other than the source tree. Here we give a sample configure script that will enable MUELU and all of its optional dependencies:

Dependency	Required		Optional	
	Library	Testing	Library	Testing
AMESOS			x	x
AMESOS2			x	x
AZTECOO				x
BELOS				x
EPETRA			x	x
IFPACK			x	x
IFPACK2			x	x
ISORROPIA			x	x
GALERI				x
KOKKOSCLASSIC			x	
TEUCHOS	x	x		
TPETRA			x	x
XPETRA	x	x		
ZOLTAN			x	x
ZOLTAN2			x	x
Boost			x	
BLAS	x	x		
LAPACK	x	x		
MPI			x	x
SuperLU 4.3			x	x

Table 2.1. MUELU's required and optional dependencies. Dependencies are further subdivided by whether the MUELU library itself has a dependency (*Library*), or whether a MUELU test has a dependency (*Testing*).

```
export TRILINOS_HOME=/path/to/your/Trilinos/source/directory
cmake -D BUILD_SHARED_LIBS:BOOL=ON \
      -D CMAKE_BUILD_TYPE:STRING="RELEASE" \
      -D CMAKE_CXX_FLAGS:STRING="-g" \
      -D Trilinos_ENABLE_EXPLICIT_INSTANTIATION:BOOL=ON \
      -D Trilinos_ENABLE_TESTS:BOOL=OFF \
      -D Trilinos_ENABLE_EXAMPLES:BOOL=OFF \
      -D Trilinos_ENABLE_MueLu:BOOL=ON \
      -D MueLu_ENABLE_TESTS:STRING=ON \
      -D MueLu_ENABLE_EXAMPLES:STRING=ON \
      -D TPL_ENABLE_BLAS:BOOL=ON \
      -D TPL_ENABLE_MPI:BOOL=ON \
      ${TRILINOS_HOME}
```

More configure examples can be found in `Trilinos/sampleScripts`. For more information on configuring, see the Trilinos quick start guide [1].

2.6 Simple Example

The most common scenario for MUELU is that the user needs an iterative linear solver with an AMG preconditioner. When using TRILINOS the user has the choice between TPETRA and EPETRA for the underlying linear algebra. For linear solvers TRILINOS provides the packages AZTECOO and BELOS which both implement the most important iterative Krylov subspace methods such as CG and GMRES.

2.6.1 MUELU as preconditioner within BELOS

Assuming that TPETRA is used for the linear algebra with a linear solver from the BELOS package the following code shows the basic steps how to use a MUELU multigrid preconditioner. The focus is on the algorithmic outline of setting up a linear solver, such that we skip the template parameters to keep the example short and clear. The user may refer to the corresponding source files within the `examples` and `test` folders for concrete examples.

First we create the MUELU multigrid preconditioner using xml parameters from a file on the hard disk (e.g., *mueluOptions.xml* in the example below).

```
1  Teuchos::RCP<Tpetra::CrsMatrix<> > A;
2  // create A here ...
3  Teuchos::RCP<MueLu::TpetraOperator> mueLuPreconditioner;
4  std::string optionsFile = "mueluOptions.xml";
5  mueLuPreconditioner = MueLu::CreateTpetraPreconditioner(A, optionsFile);
```

The xml file defines the multigrid preconditioner. A typical parameter list file for MUELU looks like

```
<ParameterList name="MueLu">

  <Parameter name="verbosity" type="string" value="low"/>

  <Parameter name="max levels" type="int" value="3"/>
  <Parameter name="coarse: max size" type="int" value="10"/>

  <Parameter name="multigrid algorithm" type="string" value="sa"/>

  <!-- Smoothing -->
  <!-- Comment/uncomment different sections to try different smoothers -->

  <!-- Jacobi -->
  <Parameter name="smoother: type" type="string" value="RELAXATION"/>
  <ParameterList name="smoother: params">
    <Parameter name="relaxation: type" type="string" value="Jacobi"/>
    <Parameter name="relaxation: sweeps" type="int" value="1"/>
    <Parameter name="relaxation: damping factor" type="double" value="0.9"/>
  </ParameterList>

  <!-- Aggregation -->
  <Parameter name="aggregation: type" type="string" value="uncoupled"/>
  <Parameter name="aggregation: min agg size" type="int" value="3"/>
  <Parameter name="aggregation: max agg size" type="int" value="9"/>

</ParameterList>
```

It defines a 3 level smoothed aggregation multigrid algorithm (optimal for symmetric positive definite matrices). The aggregation size is between 3 and 9 nodes which may be a good choice for a 2D problem. As level smoother one sweep with a damped Jacobi method is used. On the coarsest level a direct solver is applied per default. A complete list of all available parameters and valid parameter choices is given in §4 of this user guide.

Beside of the linear operator A we als need an initial guess vector for the solution and a right hand side vector for solving a linear system

```
    Teuchos::RCP<const Tpetra::Map<> > map = A->getDomainMap();

    // Create initial vectors
    Teuchos::RCP<Tpetra::MultiVector<> > B, X;
    X = Teuchos::rcp( new Tpetra::MultiVector<>(map,numrhs) );
    Belos::MultiVecTraits<>::MvRandom( *X );
    B = Teuchos::rcp( new Tpetra::MultiVector<>(map,numrhs) );
    Belos::OperatorTraits<>::Apply( *A, *X, *B );
    Belos::MultiVecTraits<>::MvInit( *X, 0.0 );
```

To generate a dummy example above code first declares to vectors. The right hand side vector is calculated as matrix vector product of a random vector with the operator A. The initial guess is finally initialized with zeros.

Then, one can define a `Belos::LinearProblem` object where the `mueLuPreconditioner` is used for left preconditioning.

```
Belos::LinearProblem<> problem( A, X, B );
problem->setLeftPrec(mueLuPreconditioner);
bool set = problem.setProblem();
```

Next, we can set up a BELOS solver using some basic parameters

```
Teuchos::ParameterList belosList;
belosList.set( "Block Size", 1 );
belosList.set( "Maximum Iterations", 100 );
belosList.set( "Convergence Tolerance", 1e-10 );
belosList.set( "Output Frequency", 1 );
belosList.set( "Verbosity", Belos::TimingDetails + Belos::FinalSummary );

Belos::BlockCGSolMgr<> solver( rcp(&problem,false), rcp(&belosList,false) );
```

Finally, one can perform the solution process using

```
Belos::ReturnType ret = solver.solve();
```

2.6.2 MUELU as preconditioner within AZTECOO

When using EPETRA the AZTECOO is an alternative for BELOS which provides fast and mature implementations of iterative linear solvers (even though the user is recommended to use the more modern BELOS implementations).

Assuming that the linear operator is given as an EPETRA object the MUELU preconditioner can be generated via

```
Teuchos::RCP<Epetra_CrsMatrix> A;
// create A here ...
Teuchos::RCP<MueLu::EpetraOperator> mueLuPreconditioner;
std::string optionsFile = "mueluOptions.xml";
mueLuPreconditioner = MueLu::CreateEpetraPreconditioner(A, optionsFile);
```

The file format for the xml parameter file is the same as for the example from §2.6.1.

Furthermore, we assume that a right hand side vector and a solution vector with the initial guess are defined

```
Teuchos::RCP<const Epetra_Map> map = A->DomainMap();
Teuchos::RCP<Epetra_Vector> B = Teuchos::rcp(new Epetra_Vector(map));
Teuchos::RCP<Epetra_Vector> X = Teuchos::rcp(new Epetra_Vector(map));
X->PutScalar(0.0);
```

Then, a Epetra_LinearProblem can be defined by

```
Epetra_LinearProblem epetraProblem(A.get(), X.get(), B.get());
```

With the following lines an AZTECOO CG solver is generated

```
AztecOO aztecSolver(epetraProblem);
aztecSolver.SetAztecOption(AZ_solver, AZ_cg);
aztecSolver.SetPrecOperator(mueLuPreconditioner.get());
```

Finally, the linear system is solved via

```
int maxIts = 100;
double tol = 1e-10;
aztecSolver.Iterate(maxIts, tol);
```

2.6.3 Further remarks

This section is only meant to give a rough overview on how to use MUELU as preconditioner within the TRILINOS packages for iterative solvers. There are other more complicated ways to use MUELU as preconditioners for BELOS and AZTECOO through the XPETRA interface. Of course, MUELU can also work as standalone multigrid solver. For more information on these topics with examples the reader may refer to the examples and tests in the MUELU source folder as well as to the MUELU tutorial ([13]).

Chapter 3

Performance tips

This Section gives few tips on tuning MUELU performance.

3.1 Tips for impatient user

1. Use matrix rebalancing options when running in parallel. See §**??**.
2. Adjust aggregation strategy. See §**??**.
3. Try replacing direct solver with a few smoothing steps, if coarse level solve becomes too expensive. See §4.6.
4. Choose a smoother whose computational kernel is a matvec, such as the Chebyshev polynomial smoother, if a problem is symmetric positive definite. See §4.4.

Chapter 4

MUELU options

In this section, we report the complete list of MUELU input parameters. It is important to notice, however, that MUELU relies on other TRILINOS packages to provide support for some of its algorithms. For instance, IFPACK/IFPACK2 provide standard smoothers like Jacobi, Gauss-Seidel or Chebyshev, while AMESOS/AMESOS2 provide access to direct solvers. The parameters affecting the behaviour of delegated algorithms are simply passed by MUELU to a routine from the corresponding package. Please consult corresponding packages for a full list of supported algorithms and corresponding parameters.

4.1 Using parameters on individual levels

Some of the parameters that affect the preconditioner can in principle be different from level to level. By default, parameters affects all levels in the multigrid hierarchy.

The settings on a particular levels can be changed by using level sublists. Level sublist is a `ParameterList` sublist with a name "level XX". The parameter names in the sublist do not require any modifications. For example, the following fragment of code

```
<ParameterList name="level 2">
  <Parameter name="smoother: type" type="string" value="CHEBYSHEV"/>
</ParameterList>
```

changes the smoother for level 2 to be a polynomial smoother.

4.2 Parameter validation

By default, MUELU validates the input parameter list. A parameter that is misspelled or unknown, or has an incorrect value type will cause an exception to be thrown and execution to halt.

☛ Spaces are important within a parameter's name. Please separate words by just one space, and make sure there are no leading or trailing spaces.

The option `print initial parameters` prints the initial list given to the interpreter. The option `print unused parameters` prints the list of unused parameters.

4.3 General options

`verbosity`	[string] Control of the amount of printed information. Possible values: "none", "low", "medium", "high", "extreme". **Default:** "high".
`number of equations`	[int] Number of PDE equations at each gride node. Only constant block size is considered. **Default:** 1.
`max levels`	[int] Maximum number of levels. **Default:** 10.
`cycle type`	[string] Multigrid cycle type. Possible values: "V", "W". **Default:** "V".
`problem: symmetric`	[bool] Symmetry of a problem. **Default:** true.

4.4 Smoothing and coarse solver options

MUELU relies on other TRILINOS packages to provide level smoothers and coarse solvers. IFPACK and IFPACK2 provide standard smoothers (see Table 4.1), and AMESOS and AMESOS2 provide direct solvers (see Table 4.2). Note that it is completely possible to use any level smoother as a direct solver.

MUELU checks parameters `smoother: * type` and `coarse: type` to determine:

- what package to use (i.e., is it a smoother or direct solver);
- possibly transform the type (in case of a smoother)

  ☛ IFPACK and IFPACK2 use different types to construct smoothers (e.g., "point relaxation stand-alone" vs "RELAXATION"). MUELU follows IFPACK2 notation for smoother types. Please consult IFPACK2 manual [10].

The parameter lists `smoother: * params` and `coarse: params` are passed directly to the corresponding package without any examination of their content. Please consult corresponding manuals for a list of possible values.

By default, MUELU uses one sweep of symmetric Gauss-Seidel for both pre- and post-smoothing, and SuperLU for coarse system solver.

`smoother: type`	
`RELAXATION`	Point relaxation smoothers, including Jacobi, Gauss-Seidel, symmetric Gauss-Seidel, etc. The exact smoother is chosen by specifying `relaxation: type` parameter in the `smoother: params` sublist.
`CHEBYSHEV`	Chebyshev polynomial smoother.
`ILUT`, `RILUK`	Local (processor-based) incomplete factorization methods.

Table 4.1. Commonly used smoothers provided by IFPACK/IFPACK2. Note that these smoothers can also be used as coarse grid solvers.

`coarse: type`	AMESOS	AMESOS2	
`KLU`	x		Default AMESOS solver [7].
`KLU2`		x	Default AMESOS2 solver [4].
`SuperLU`	x	x	Third-party serial sparse direct solver [9].
`SuperLU_dist`	x	x	Third-party parallel sparse direct solver [9].
`Umfpack`	x		Third-party solver [6].
`Mumps`	x		Third-party solver [2].

Table 4.2. Commonly used direct solvers provided by AMESOS/AMESOS2

`smoother: pre or post` [string] Smoother combination. Possible values: "pre", "post", "both", "none". **Default:** "both".

`smoother: type` [string] Smoother type. Possible values: see Table 4.1. **Default:** one sweep of symmetric Gauss-Seidel.

`smoother: pre type`	[string] Pre-smoother type. Possible values: see Table 4.1. **Default:** one sweep of symmetric Gauss-Seidel.
`smoother: post type`	[string] Post-smoother type. Possible values: see Table 4.1. **Default:** one sweep of symmetric Gauss-Seidel.
`smoother: params`	[ParameterList] Smoother parameters. For standard smoothers, MUELU passes them directly to STRATIMIKOS.
`smoother: pre params`	[ParameterList] Pre-smoother parameters. For standard smoothers, MUELU passes them directly to STRATIMIKOS.
`smoother: post params`	[ParameterList] Post-smoother parameters. For standard smoothers, MUELU passes them directly to STRATIMIKOS.
`smoother: overlap`	[int] Smoother subdomain overlap. **Default:** 0.
`smoother: pre overlap`	[int] Pre-smoother subdomain overlap. **Default:** 0.
`smoother: post overlap`	[int] Post-smoother subdomain overlap. **Default:** 0.
`coarse: max size`	[int] Maximum dimension of the coarse grid. MUELU will stop coarsening once it is achieved. **Default:** 2000.
`coarse: type`	[string] Coarse solver. Possible values: see Table 4.2. **Default:** "SuperLU".
`coarse: params`	[ParameterList] Coarse solver parameters. MUELU passes them directly to coarse solver.
`coarse: overlap`	[int] Coarse solver subdomain overlap. **Default:** 0.

4.5 Aggregation options

`uncoupled`	Attempts to construct aggregates of optimal size (3^d nodes in d dimensions). Each process works independently, and aggregates cannot span processes.
`coupled`	Attempts to construct aggregates of optimal size (3^d nodes in d dimensions). Aggregates are allowed to cross processor boundaries. Use carefully. If unsure use `uncoupled` instead.

Table 4.3. Available coarsening schemes.

`aggregation:`	`type`	[string] Aggregation scheme. Possible values: "uncoupled", "coupled". **Default:** "uncoupled".
`aggregation:`	`ordering`	[string] Ordering strategy. Possible values: "natrual", "graph", "random". **Default:** "natural".
`aggregation:`	`drop scheme`	[string] Aggregation connectivity dropping scheme. Possible values: "classical", "distance laplacian". **Default:** "classical".
`aggregation:`	`drop tol`	[double] Aggregation dropping threshold. **Default:** 0.0.
`aggregation:`	`min agg size`	[int] Minimum size of an aggregate. **Default:** 2.
`aggregation:`	`max agg size`	[int] Maximum size of an aggregate. **Default:** 2147483647.
`aggregation:`	`Dirichlet threshold`	[double] Threshold for determining whether entries are zero during Dirichlet row detection. **Default:** 0.0.
`aggregation:`	`export visualization data`	[bool] Export data for visualization post-processing. **Default:** false.

4.6 Rebalancing options

`repartition:`	`enable`	[bool] Repartitioning on/off switch. **Default:** false.
`repartition:`	`partitioner`	[string] Partitioning package to use. Possible values: "zoltan", "zoltan2". **Default:** "zoltan2".
`repartition:`	`params`	[`ParameterList`] Partitioner parameters. MUELU passes them directly to partitioner.
`repartition:`	`start level`	[int] Minimum level to run partitioner. MUELU does not repartition for finer levels. **Default:** 2.
`repartition:`	`min rows per proc`	[int] Desired minimum number of rows per processor. If actual number if smaller, then repartitioning occurs. **Default:** 800.
`repartition:`	`max imbalance`	[double] Desired maximum nonzero imbalance ratio. **Default:** 1.2.
`repartition:`	`remap parts`	[bool] Postprocessing for partitioning to reduce data migration. **Default:** true.
`repartition:`	`rebalance P and R`	[bool] Do rebalancing of R and P during the setup. This speeds up the solve, but slows down the setup phases. **Default:** true.

4.7 Multigrid algorithms

`multigrid algorithm`	[string] Multigrid method. Possible values: "unsmoothed", "sa", "emin", "pg". **Default:** "sa".

`semicoarsen: coarsen rate`	[int] Rate at which to coarsen unknowns in the z direction. **Default:** 3.
`sa: damping factor`	[double] Damping factor for smoothed aggregation. **Default:** 1.33333333.
`sa: use filtered matrix`	[bool] Matrix to use for smoothing the tentative prolongator. The two options are: to use the original matrix, and to use the filtered matrix with filtering based on filtered graph used for aggregation. **Default:** true.
`filtered matrix: use lumping`	[bool] During construction of a filtered matrix, we have an option to add dropped entries to the diagonal. This is useful for preserving constant nullspace for the Laplacian type matrix. **Default:** true.
`filtered matrix: reuse eigenvalue`	[bool] During construction of a filtered matrix, we have an option to get the eigenvalue estimate from the original matrix. This allows us to skip heavy computation. **Default:** true.
`emin: iterative method`	[string] Iterative method to use for energy minimization of intial prolongator in energy-minimization. Possible values: "cg", "sd". **Default:** "cg".
`emin: num iterations`	[int] Number of iterations to minimize initial prolongator energy in energy-minimization. **Default:** 2.
`emin: num reuse iterations`	[int] Number of iterations to minimize the reused prolongator energy in energy-minimization. **Default:** 1.
`emin: pattern`	[string] Sparsity pattern to use for energy minization. Possible values: "AkPtent". **Default:** "AkPtent".
`emin: pattern order`	[int] Matrix order for the "AkPtent" pattern. **Default:** 1.

4.8 Miscellaneous options

`export data`	[`ParameterList`] Exporting a subset of the hierarchy data in a file. Currently, the list can contain any of three parameter names (”A”, ”P”, ”R”) of type ”string” and value ”{levels separated by commas}”. A matrix is saved in two files: a) data is saved in the MatrixMarket format in a file called ”A_level.mm”, or similar; b) row map is saved in the MatrixMarket format in a file called ”rowmap_A_level.mm”, or similar.
`print initial parameters`	[bool] Print parameters provided for a hierarchy construction. **Default:** true.
`print unused parameters`	[bool] Print parameters unused during a hierarchy construction. **Default:** true.
`transpose: use implicit`	[bool] Use implicit transpose for the restriction operator. **Default:** false.

References

[1] Trilinos cmake quickstart. `http://trilinos.org/build_instructions.html`, 2014.

[2] Patrick R Amestoy, Iain S Duff, Jean-Yves LExcellent, and Jacko Koster. Mumps: a general purpose distributed memory sparse solver. In *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, pages 121–130. Springer, 2001.

[3] Roscoe A Bartlett. Teuchos:: RCP beginners guide. Technical Report SAND2004-3268, Sandia National Labs, 2010.

[4] Eric Bavier, Mark Hoemmen, Sivasankaran Rajamanickam, and Heidi Thornquist. Amesos2 and belos: Direct and iterative solvers for large sparse linear systems. *Scientific Programming*, 20(3):241–255, 2012.

[5] William L Briggs, Steve F McCormick, et al. *A multigrid tutorial*. SIAM, 2nd edition, 2000.

[6] Timothy A Davis. Algorithm 832: Umfpack v4. 3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):196–199, 2004.

[7] Timothy A Davis and Ekanathan Palamadai Natarajan. Algorithm 907: Klu, a direct sparse solver for circuit simulation problems. *ACM Transactions on Mathematical Software (TOMS)*, 37(3):36, 2010.

[8] M.W. Gee, C.M. Siefert, J.J. Hu, R.S. Tuminaro, and M.G. Sala. ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[9] Xiaoye S. Li, James W. Demmel, John R. Gilbert, Laura Grigori, Meiyue Shao, and Ichitaro Yamazaki. SuperLU Users' Guide. 2011.

[10] Christopher Siefert and Jonathan Hu. Ifpack2 Users Guide. Technical report, Sandia National Labs, 2014.

[11] Heidi Thornquist, Ross Bartlett, Mark Hoemmen, Christopher Baker, and Michael Heroux. Teuchos: Trilinos tools library. `http://trilinos.org/packages/teuchos`, 2014.

[12] Ulrich Trottenberg, Cornelis Oosterlee, and Anton Schuller. *Multigrid*. Elsevier Academic Press, 2001.

[13] Tobias A. Wiesner, Michael W. Gee, Andrey Prokopenko, and Jonathan Hu. The MueLu tutorial. `http://trilinos.org/packages/muelu`, 2014. SAND2014-18624R.

DISTRIBUTION:

1 Tobias Wiesner
Institute for Computational Mechanics
Technische Universität München
Boltzmanstraße 15
85747 Garching, Germany

1 MS 1320 Michael Heroux, 1446
1 MS 1318 Robert Hoekstra, 1446
1 MS 1320 Mark Hoemmen, 1446
1 MS 1320 Paul Lin, 1446
1 MS 1318 Andrey Prokopenko, 1426
1 MS 1322 Christopher Siefert, 1443
1 MS 0899 Technical Library, 9536 (electronic copy)

DISTRIBUTION:

1	MS 9159	Jonathan Hu, 1426
1	MS 9159	Paul Tsuji, 1442
1	MS 9159	Raymond Tuminaro, 1442
1	MS 0899	Technical Library, 8944 (electronic copy)

Sandia National Laboratories